CATERVA

A Compressed And Multidimensional Container
For Not So Big Data

@FrancescAlted
Freelancer

Sponsored by NUMFOCUS OPEN CODE = BETTER SCIENCE

A Member of the BLOSC Family

HDF5 European Workshop. Grenoble September 18th, 2019

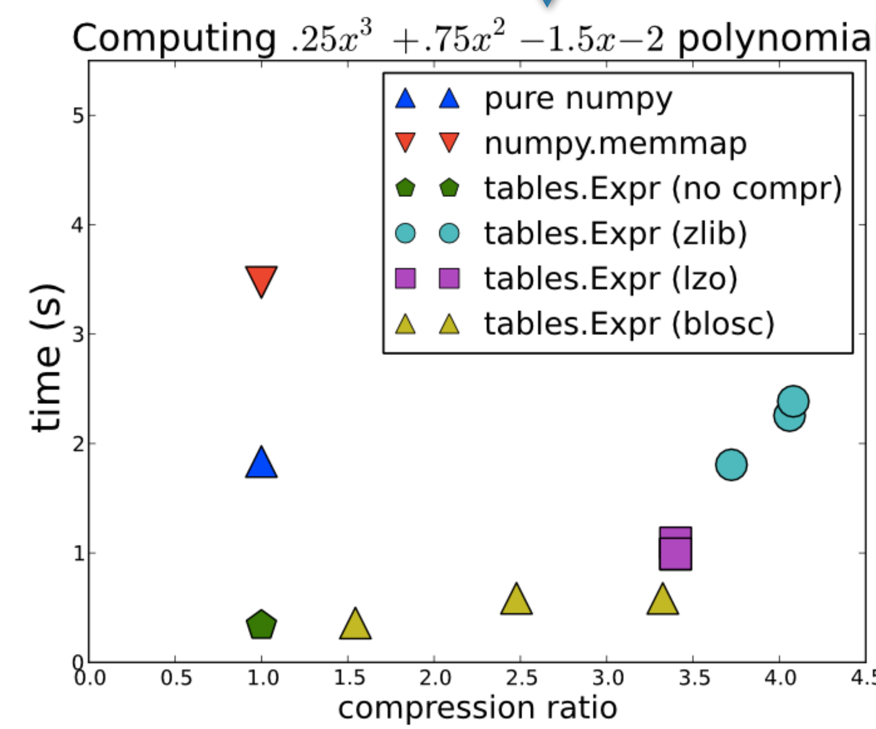
About Me

- Physicist by training.
- Computer scientist by passion.
- Open Source enthusiast by philosophy.
  - PyTables (2002 - 2011)
  - Blosc (2009 - now)

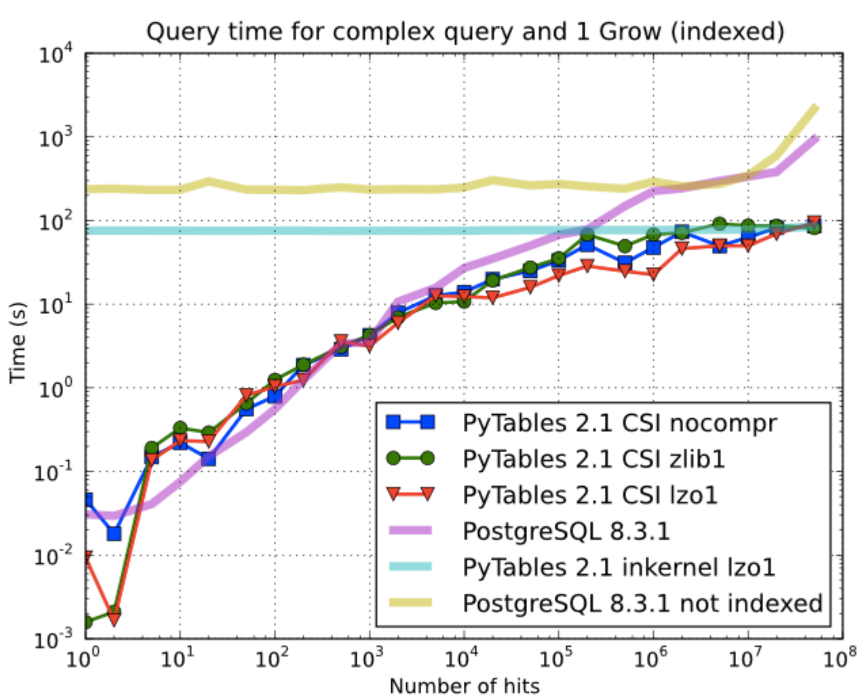
PyTables

Out-of-core
Expressions

The technology platform to make a difference in your relationship with large and complex data

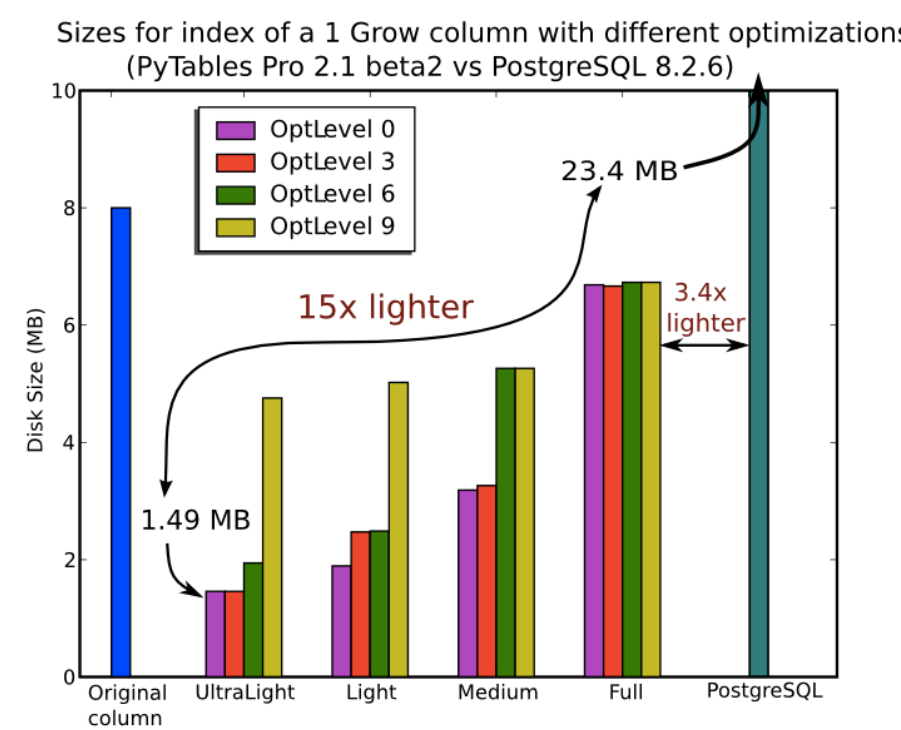


HDF5 + a Twist



OPSI

Indexed
Queries



What is Caterva?

- It is an open source **C library and a format** that allows to store large multidimensional, chunked, compressed datasets.

- Data can be stored either **in-memory or on-disk**, but the API to handle both versions is the same.

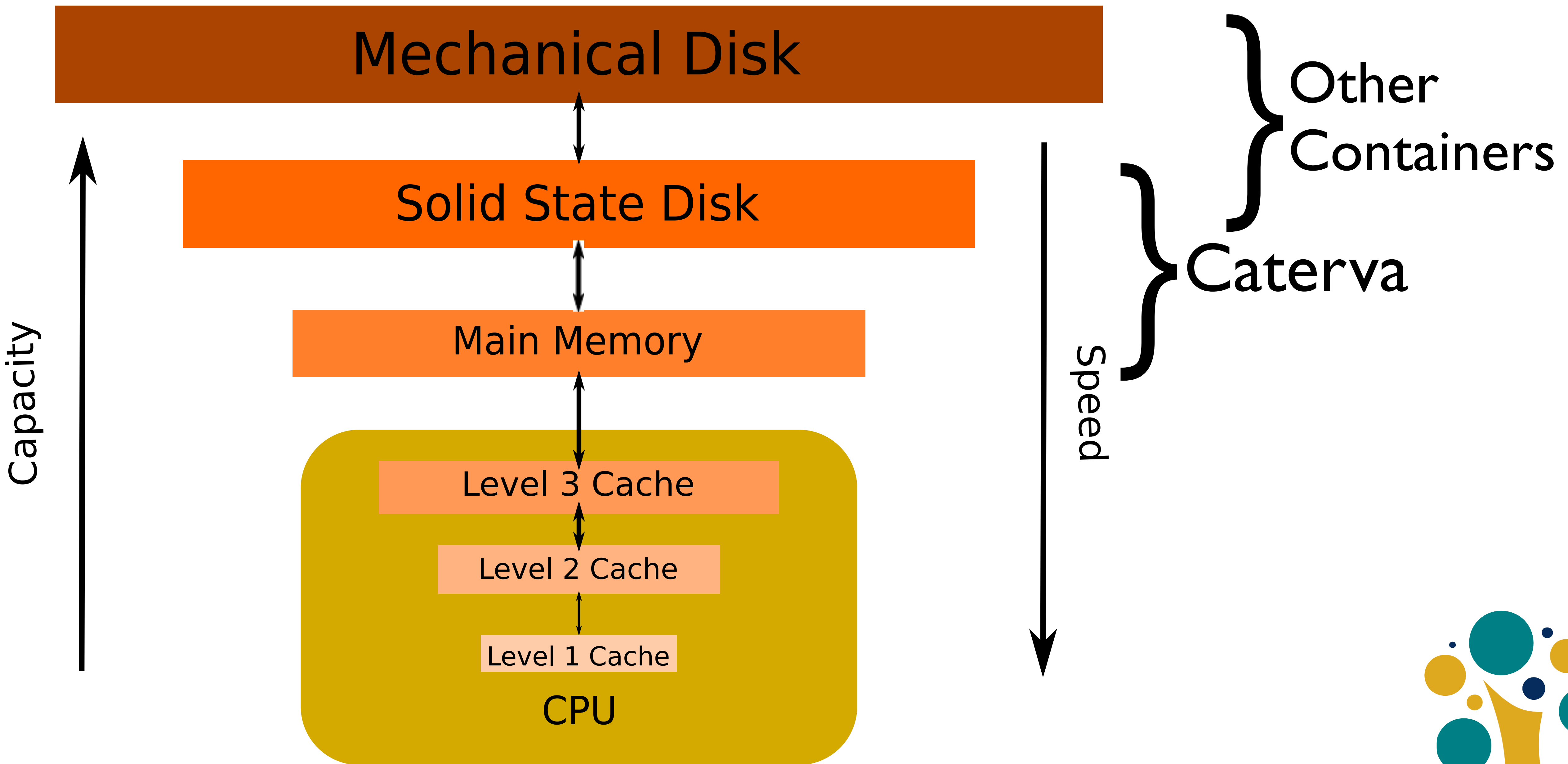
- Compression is handled transparently for the user by adopting the **Blosc2 library**.

Sponsored by NUMFOCUS
OPEN CODE = BETTER SCIENCE

Why Another Data Container?

- Most of the existing data containers supporting **on-the-flight compression** are meant for on-disk/cloud data.
- But the memory layer can be seen as storage too, and there is a **need for a container** that is optimized for this.
- Caterva is **designed from the ground up** to use the memory layer as storage for a compressed data-container.

Accelerating I/O With Caterva



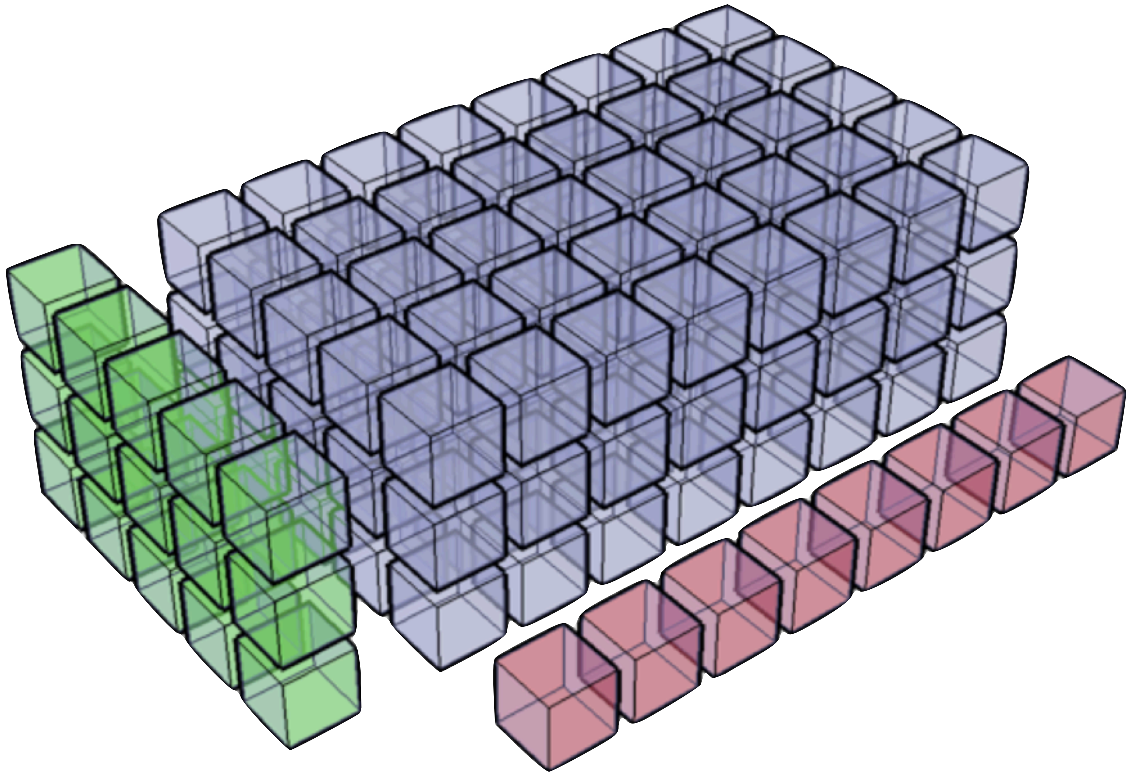
Why Another Format?

- Being able to store in an in-memory data container does not mean that data cannot be persisted. It is **critical** to find a way to **store and retrieve data efficiently**.
- Also, it is important to adopt **open formats** for reducing the maintenance burden and facilitate its adoption more quickly.
- As we will see soon, Caterva brings an **efficient** and **open format** for persistency.

Caterva Brings Powerful Slicing Capabilities

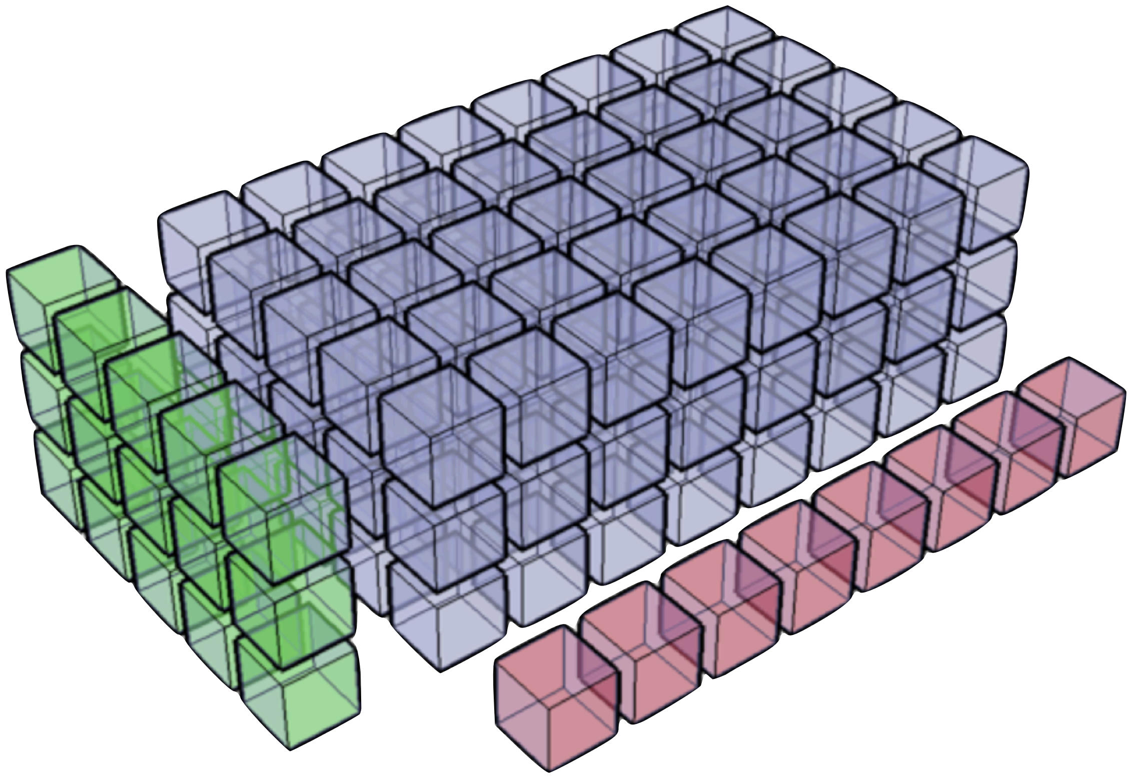
- Caterva's main feature is to be able to extract all kind of slices out of high dimensional datasets, efficiently.

- Resulting slices can be either Caterva containers or regular plain buffers (for better interaction with e.g. NumPy).



Accessing Chunked Datasets

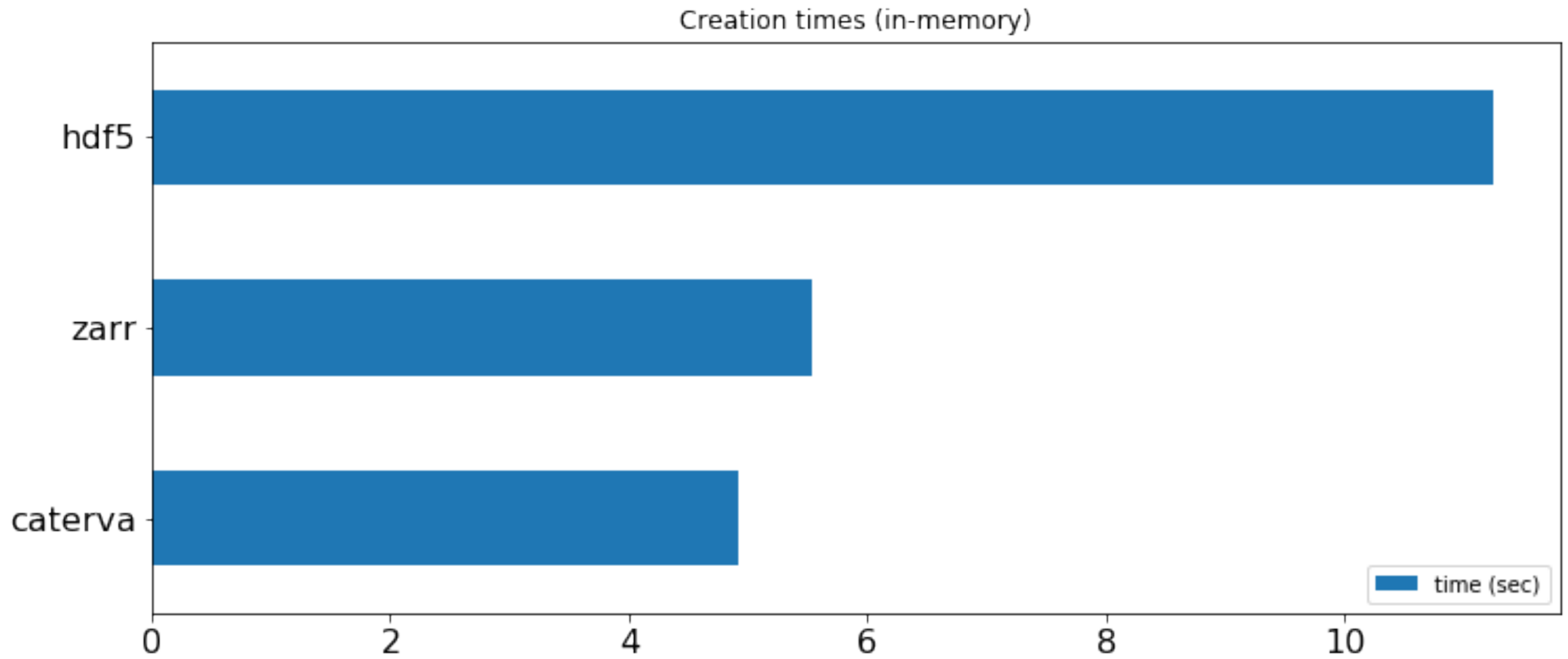
- Those used to manipulate chunked multidimensional arrays know how critical choosing the partition size is.

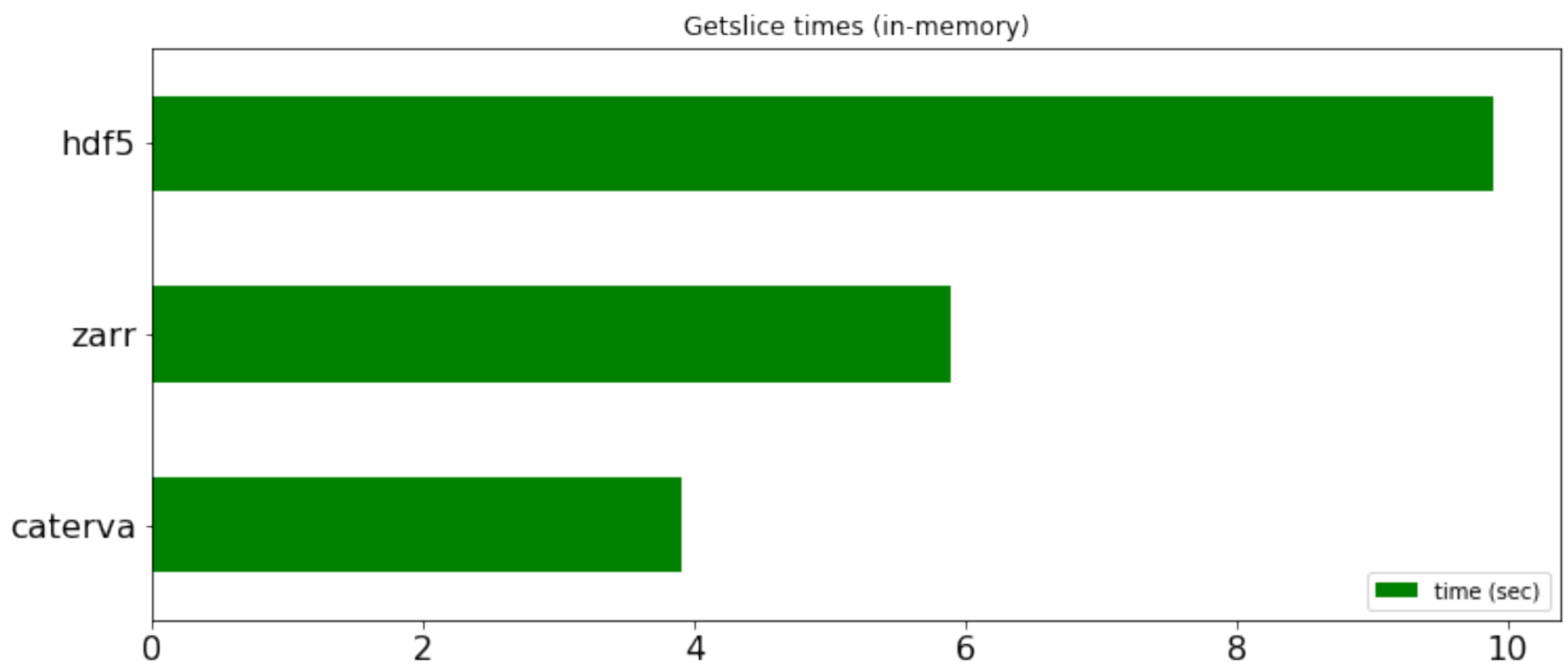


You can play with a small, but representative benchmark at:

https://github.com/Blosc/cat4py/blob/master/notebooks/compare_getslice.ipynb

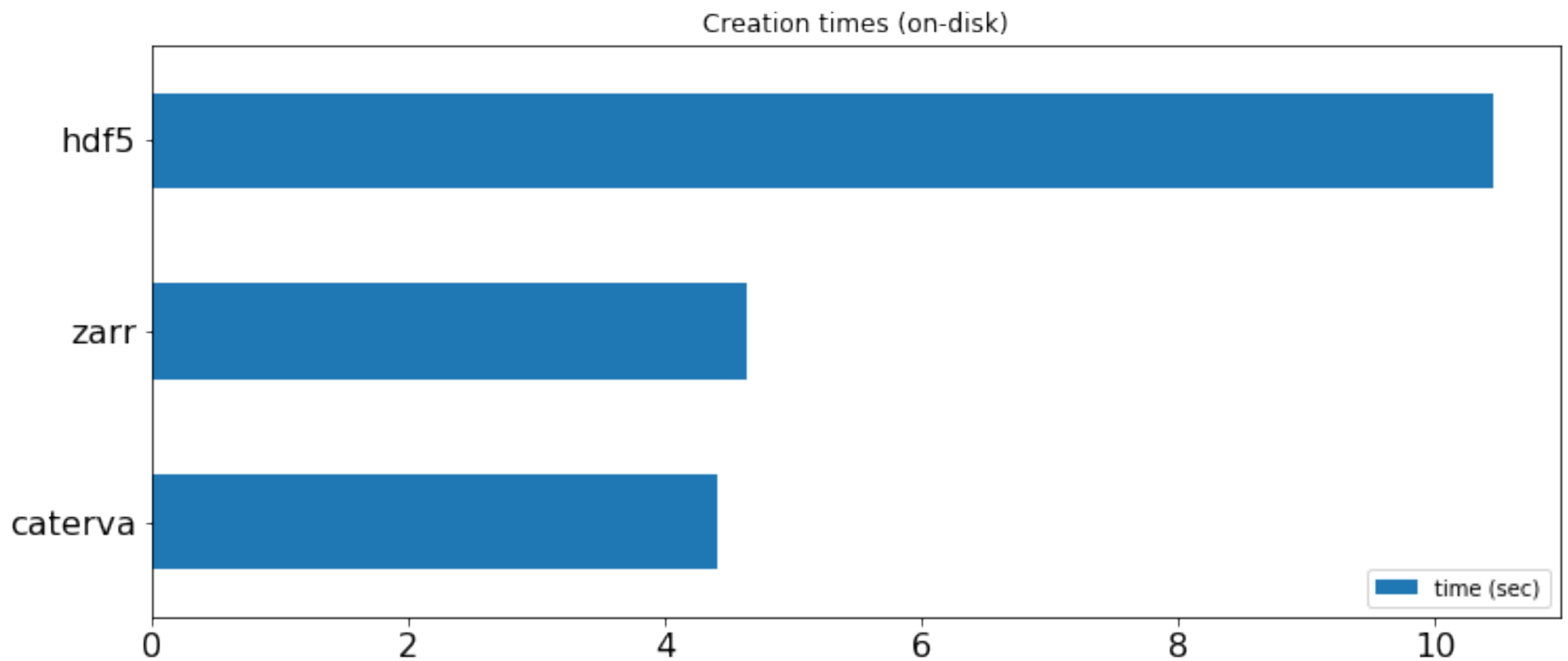
Performance In-Memory

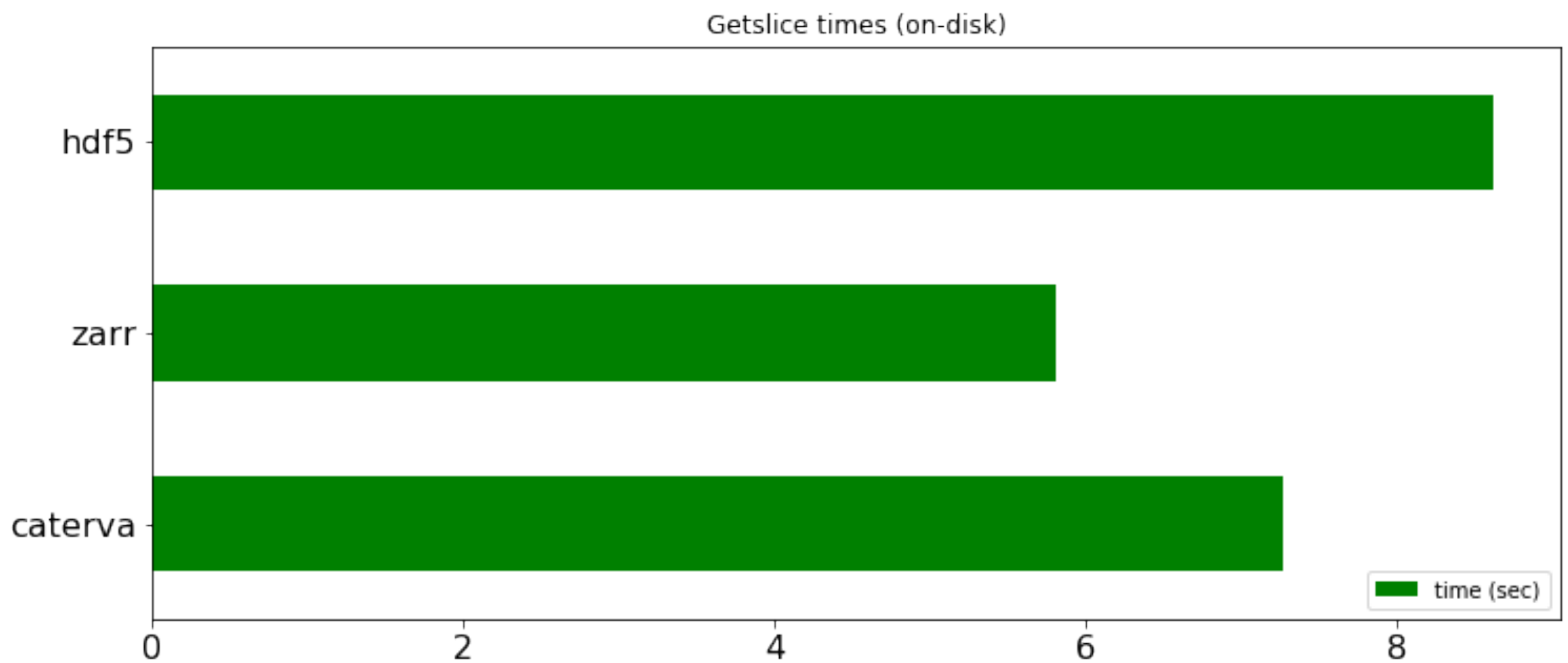




Caterva is meant to read data from memory very fast!

Performance On-Disk





There is still room for optimization when reading from disk…

```c
#include <caterva.h>

int main(){
    // Create a context
    caterva_ctx_t *ctx = caterva_new_ctx(NULL, NULL, BLOSC2_CPARAMS_DEFAULTS, BLOSC2_DPARAMS_DEFAULTS);
    ctx->cparams.typesize = sizeof(double);

    // Define the partition shape for the first array
    int8_t ndim = 3;
    int64_t pshape_[] = {3, 2, 4};
    caterva_dims_t pshape = caterva_new_dims(pshape_, ndim);

    // Create the first (empty) array
    caterva_array_t *cat1 = caterva_empty_array(ctx, NULL, &pshape);

    // Define a buffer shape to fill cat1
    int64_t shape_[] = {10, 10, 10};
    caterva_dims_t shape = caterva_new_dims(shape_, ndim);

    // Create a buffer to fill cat1
    size_t buf1size = 10 * 10 * 10 * sizeof(double);
    double *buf1 = (double *) malloc(buf1size * sizeof(double));

    // Fill cat1 with the above buffer
    caterva_from_buffer(cat1, &shape, buf1);

    free(buf1);
    caterva_free_array(cat1);

    return 0;
}
```

Example of muti-dimensional array creation

```c
// Apply a `get_slice` to cat1 and store it into cat2
int64_t start_[] = {3, 6, 4};
caterva_dims_t start = caterva_new_dims(start_, ndim);
int64_t stop_[] = {4, 9, 8};
caterva_dims_t stop = caterva_new_dims(stop_, ndim);

int64_t pshape2_[]  = {1, 2, 3};
caterva_dims_t pshape2 = caterva_new_dims(pshape2_, ndim);
caterva_array_t *cat2 = caterva_empty_array(ctx, NULL, &pshape2);

caterva_get_slice(cat2, cat1, &start, &stop);
caterva_squeeze(cat2);

// Create a buffer to store the cat2 elements
uint64_t buf2size = 1;
caterva_dims_t shape2 = caterva_get_shape(cat2);
for (int j = 0; j < shape2.ndim; ++j) {
    buf2size *= shape2.dims[j];
}
double *buf2 = (double *) malloc(buf2size * sizeof(double));

// Fill buffer with the cat2 content
caterva_to_buffer(cat2, buf2);

printf("The resulting hyperplane is:\n");
for (int64_t i = 0; i < shape2.dims[0]; ++i) {
    for (int64_t j = 0; j < shape2.dims[1]; ++j) {
        printf("%6.f", buf2[i * cat2->shape[1] + j]);
    }
    printf("\n");
}
```

Example of getting a slice out of a muti-dimensional array

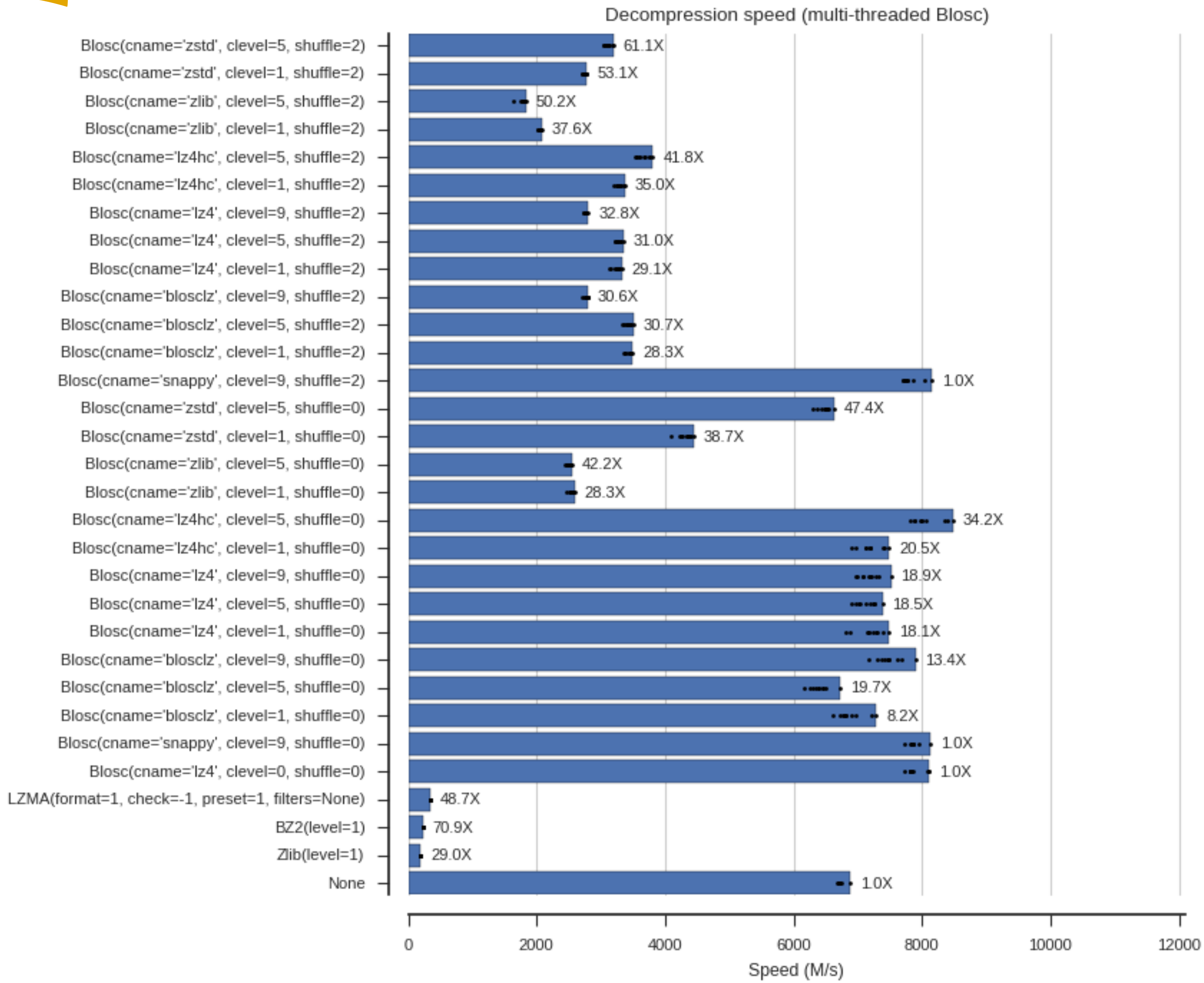
Brief Comparison Against Well Known Chunked Containers

	HDF5	Zarr	Caterva
One-file per container?	Yes (> 1 container)	No (1 file per chunk)	Yes
Hierarchical	Yes	Yes	No (use the filesystem)
Mature	Yes	Yes	In process
In-memory version?	Yes (sequential?)	Yes (sparse)	Yes (sequential / sparse)

Blosc2

- Blosc2 is the next generation of the well-known Blosc (aka Blosc1).
- New features:
    - Enlargeable 64-bit containers: in-memory or on-disk
    - New compression codecs
    - New filters
    - Metalayers
    - User metadata

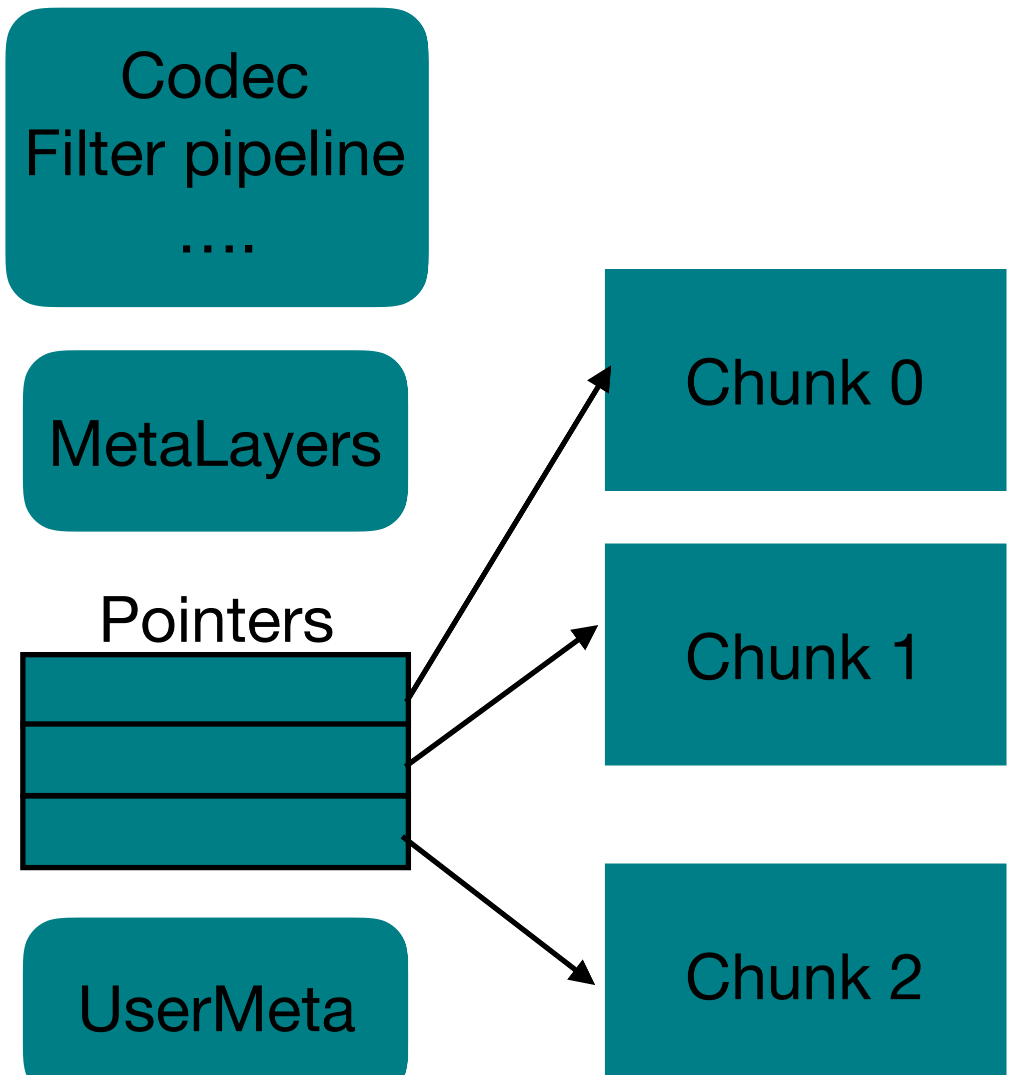
Decompression Speed



http://alimanfoo.github.io/2016/09/21/genotype-compression-benchmark.html

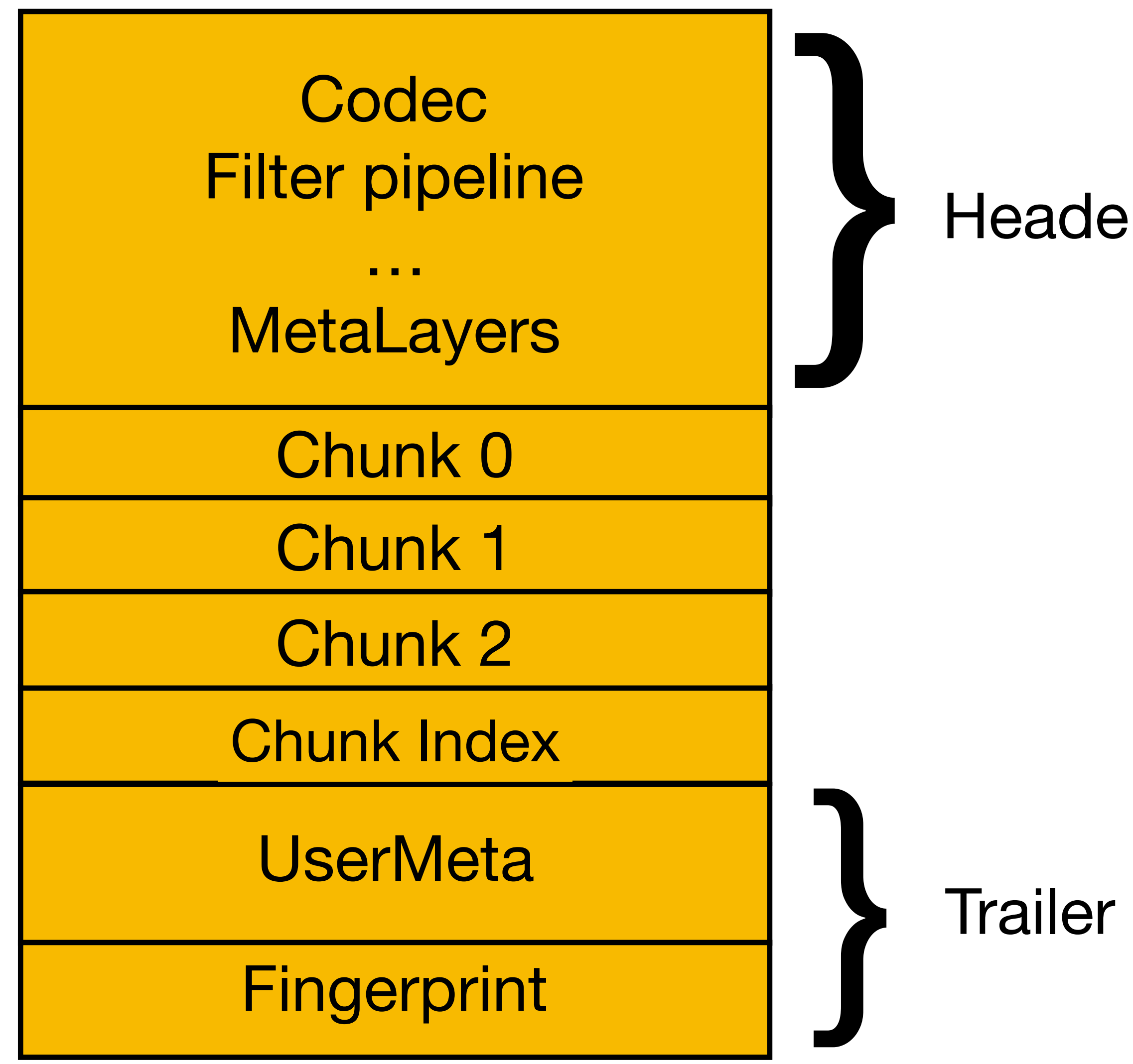
Containers in Blosc2

Super-chunk



- Sparse
- In-memory

Frame

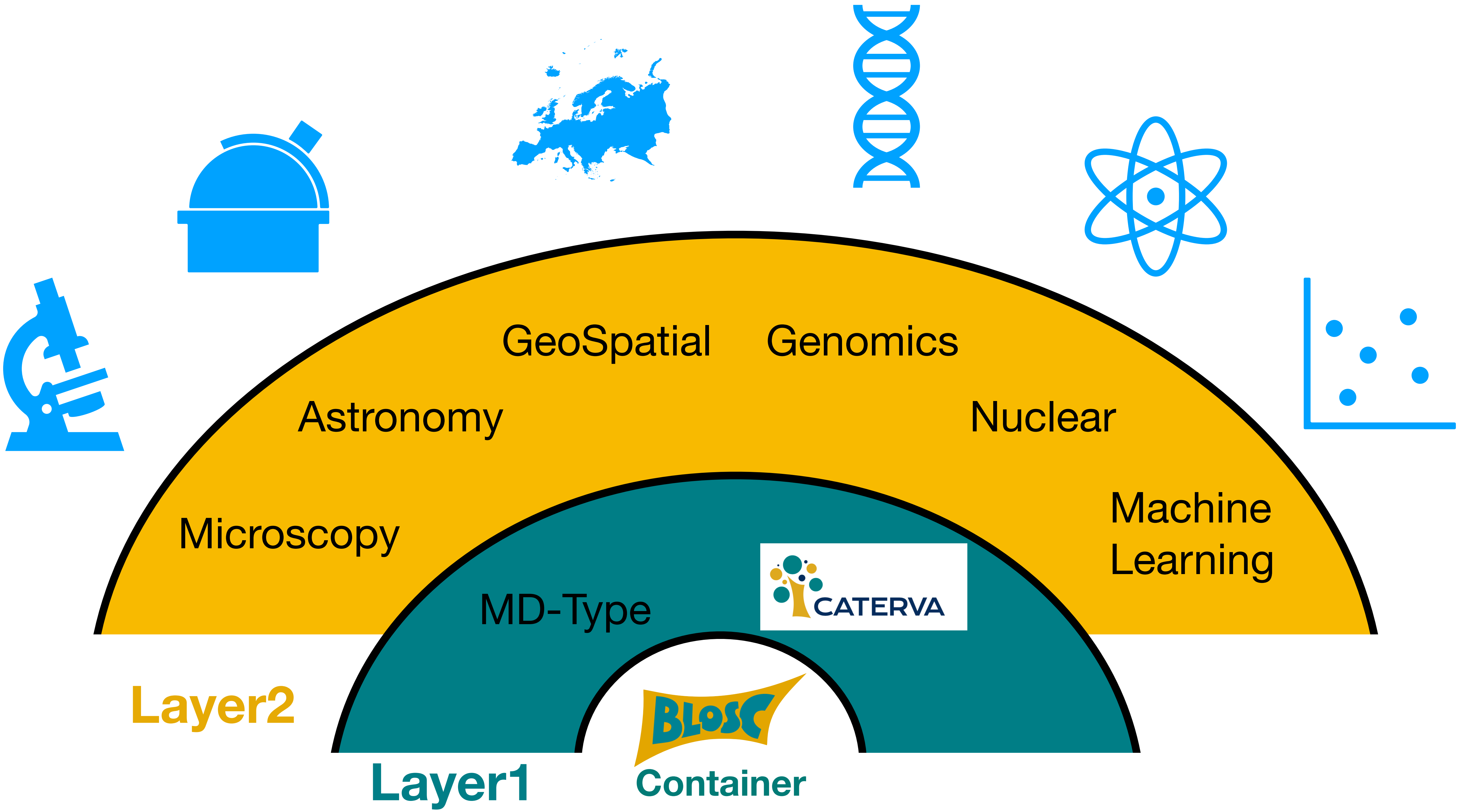


- Sequential
- In-memory / On-disk

MetaLayers in Blosc2

- Metalayers are small metadata for informing about the kind of data that is stored on a Blosc2 container.
- They are handy for defining layers with different specs: multi-dimensions, data types, geo-spatial…

MetaLayers in Blosc2



Multiple layers to target different data aspects

Caterva MetaLayer

Caterva specifies a metalayer on top of a Blosc2 container for storing multidimensional information:

```
typedef struct {
    int8_t ndim;
    //!< The number of dimensions
    uint64_t dims[CATERVA_MAXDIM];
    //!< The size of each dimension
    int32_t pdims[CATERVA_MAXDIM];
    //!< The size of each partition dimension
} caterva_dims_t;
```

This metalayer can be modified so that the shapes can be updated (e.g. an array can **grow or shrink**).

Why Caterva is Type Agnostic?

- There are too many data type systems floating around.
- Multi-dimensionality is orthogonal to the data type.
- This is why we decided not to make the type part of Caterva.
- The interested parties can always define a metalayer for endowing the desired type system to the data.

Example: add a metalayer for specifying the data type
https://github.com/Blosc/cat4py/blob/master/notebooks/array-metalayer.ipynb

Frame Format and MetaLayers Specs

- The format for a Blosc2 frame is completely specified at:
  - https://github.com/Blosc/c-blosc2/blob/master/README_FRAME_FORMAT.rst
- The format for a Caterva metalayer:
  - https://github.com/Blosc/Caterva/blob/master/README_CATERVA_METALAYER.rst

Everything specified in the msgpack format.

Where Caterva Can Help?

- Whenever there is a need to deal with **multidimensional datasets as fast as possible**.
- Provide a **backend for other packages** (bcolz? zarr?).
    - Caterva is written in portable C99, so no limitations to be wrapped from **other languages than e.g. Python**.
- Allow to create **different metalayers** that adapt to user's needs.

Where You Can Help?

- Blosc2, Caterva and cat4py (Caterva's Python wrapper), are all **open source**, so you can always **contribute with ideas and code**.

- If you like the concepts behind the **Blosc project** as a whole, and you don't have time to contribute with code, please **donate** to:

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Overview

- Caterva is a **C library and a format** for handling multidimensional data **on top of Blosc2 containers**.

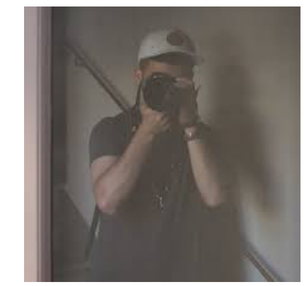
- The main goal is to efficiently **leverage fast storage** like memory, persistent memory (Intel Optane) or SSDs.

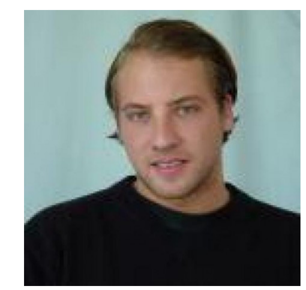
- You can use **metalayers for adapting** Caterva containers **to your own needs**.

https://github.com/Blosc/caterva

https://github.com/Blosc/c-blosc2

Acknowledgements





- First and foremost to Aleix Alcacer who contributed most of the code behind Caterva.

- Christian Steiner, for suggestions and improvements on Blosc2 / Caterva projects.

- Pepe Aracil, for his proposal for using msgpack for serializing Blosc2 containers.

- Last but not least, NumFOCUS for providing funding for developing Blosc2 and Caterva.

Thank You!

Questions?

CATERVA